# Énumérations

J. Sam, J.-C. Chappelier, M. Schinz

version 1.0

**Résumé**

Le but de ce document est de présenter la notion d'*énumération* en Java, notion utile dans ce projet pour pouvoir modéliser les états des animaux de façon adéquate.

## 1 Problématique

En programmation, il est souvent nécessaire de lier un certain nombre de constantes à un concept particulier ; par exemple, les couleurs « pique », « cœur », « carreau » et « trèfle » pour un jeu de cartes. Autre exemple : dans ce projet nous souhaitons modéliser le concept d'« état d'un animal » lequel conditionnera sa façon de se comporter. Les valeurs possibles de cet état seront « au repos », « en fuite » ou encore « en cours d'attaque ».

Quel type utiliser pour la modéliser la notion d'« état d'un animal » ?

### 1.1 Solution 1 : chaînes de caractères

Une première idée pourrait être d'utiliser le type `String` pour la notion d'état et des constantes de ce type pour les états valides, comme par exemple `"au repos"`. Cela donnerait par exemple :

```
class Animal {
    public final static String IDLE = "au repos";
    // ... idem pour ESCAPING, ATTACK, etc.

    private String state;

    public Animal(String state) {
        if (state.equals(IDLE)) {
            this.state = state;
            // ... (autres choses à faire au repos)
        }
```

```
    }
```

Cette solution n'est pourtant pas très satisfaisante car :

- le concept même d'état n'est pas très visible à la lecture du code (pas de type spécifique pour le désigner) ;
- `String` est un type trop général qui inclut énormément de valeurs invalides possibles (par exemple `"Hello world"` ou `"cucurbitacée"` !) pour décrire l'état d'un animal ; il y a en effet une *infinité* de valeurs de type `String`, mais seules *trois* d'entre-elles seraient valides pour décrire des d'états dans notre projet, ce qui implique de faire constamment des vérifications potentiellement coûteuses du genre :

```
public Animal(String direction) {
    if ( ! (   direction.equals(IDLE)
            || direction.equals(ESCAPING)
            || direction.equals(ATTACK)
           )
       )
        throw new IllegalArgumentException(...);
}
```

## 1.2 Solution 2 : objets

Une solution alternative consiste à introduire une classe pour représenter la notion d'état :

```
final class State {
    public final static State IDLE = new State("IDLE");
    // ... idem pour ESCAPING, et ATTACK

    private final String stateName;
    private State(String stateName) {
        this.stateName = stateName;
    }
}

final class Animal {
    private State state;
    //...
}
```

Vous noterez l'utilisation d'un constructeur *privé* pour ne permettre que certaines valeurs bien précises pour l'attribut représentant le nom de l'état.

Un animal serait ainsi instancié comme suit dans ce contexte :

```
Animal mouse = new Animal(State.IDLE);
```

Avec cette solution orientée objet, le constructeur de `Animal` devient plus simple car il n'y a plus besoin de vérifier ses arguments, qui sont valides par construction (si on ignore le cas `null` !) :

```
class Animal {
    private State state;

    public Animal(State state) {
        this.state = state;
    }
    //...
}
```

De plus, il devient possible d'ajouter d'autres méthodes ou des champs :

```
final class State {
    // ... comme avant
    public String frenchName() {
        if (this==IDLE) return "au repos";
        if (this==ESCAPING) return "en fuite";
        if (this==ATTACK) return "en cours d'attaque";
        return "état inconnu";
    }
}
```

La solution orientée-objets est bonne, mais néanmoins fastidieuse à mettre en oeuvre. Elle demande en effet l'écriture de beaucoup de code répétitif.

Pour éviter de devoir écrire ce code à chaque fois, Java offre la notion d'*énumération*.

# 2 Énumérations

Au moyen des énumérations, le type `State` peut se définir simplement ainsi :

```
// fichier State.java
public enum State {
    IDLE, ESCAPING, ATTACK;
}
```

Java se charge de traduire les énumérations ainsi formulées en classes encore plus complètes que celles que nous avons écrites. On peut alors déclarer et initialiser des variables de ce type énuméré par exemple comme suit :

```
State state = State.IDLE;
```

Exactement comme dans notre version orientée objet, une énumération Java est traduite en une classe où :

— chaque élément de l'énumération est traduit en une instance de cette classe ;
— ces instances sont des champs *statiques non modifiables* de la classe.

Java définit de plus quelques méthodes utiles sur les énumérations et leurs éléments :

— la méthode statique `values()`, définie sur l'énumération elle-même, retourne un tableau contenant la totalité des éléments de l'énumération, dans l'ordre de déclaration ;
par exemple, pour notre énumération `State` ci-dessus, l'appel `State.values()` retourne un tableau de trois `State`, contenant `IDLE` à la position 0, `ESCAPING` à la position 1, et `ATTACK` à la position 2 ;
— la méthode `ordinal()` retourne un entier indiquant la position de l'élément dans l'énumération (à partir de zéro) ; toujous pour notre exemple : `State.IDLE.ordinal()` retourne 0, et `State.ESCAPING.ordinal()` retourne 1 ;
— une redéfinition de `toString()` qui retourne une version chaîne de caractères de la valeur correspondante, p.ex. `"IDLE"`.

## 2.1 Utilisation de `switch`

Les éléments d'une énumération sont utilisables avec la commande `switch`. Voici un petit exemple d'utilsation de `switch` dans le contexte de notre exemple :

```
class Animal {
    private State state;
    //...
    public void move() {
        switch (state) {
        case IDLE:    //...
        case ESCAPING:  //...
        case ATTACK : //...
        default:     //...
        }
    }
}
```

Notez que dans le cas particulier des `switch`, il est possible pour les valeurs d'omettre le nom de la classe en préfixe ; p.ex. on peut écrire simplement `IDLE` au lieu de `State.IDLE`.

## 2.2 Imbrication dans une classe

Le concept d'« état d'un animal » n'a pas vraiment de raison d'être en dehors de la classe `Animal`. Il est dans notre cas préférable d'imbriquer ce type dans la classe `Animal` elle-même. Il suffit pour cela d'en placer la déclaration à l'*intérieur* de la classe, comme indiqué dans l'exemple suivant :

```
class Animal {

    public enum State {IDLE, ESCAPING, ATTACK};

    public update(...) {
        ...
        setState(State.IDLE);
        ...
    }

}
```

Si nécessaire, à l'extérieur de la classe `Animal` le type `State` doit être spécifié en le rapportant au nom de la classe auquel il appartient, c.-à-d. en écrivant `Animal.State`.

# Références

Oracle fournit le tutoriel suivant à propos des énumérations :

— *Enum types*.